ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 318 (2004) 57–78

Theoretical Computer Science

www.elsevier.com/locate/tcs

Separating NC along the δ axis

S. Bellantoni[a,*,1], I. Oitavem[b,2]

[a] *151 Howland Avenue, Toronto, Ontario, Canada M5R 3B4*

[b] *CMAF—Univ. de Lisboa, Av. Prof. Gama Pinto, 2, 1649-003 Lisboa, Portugal*

Abstract

The "δ axis" consists of certain classes N^k such that uniform $\mathrm{NC} = \bigcup_k \mathrm{N}^k$. A diagonalization is used to prove that $\mathrm{N}^k \neq \mathrm{N}^{k+1}$ for all $k \geqslant 0$. This separation of NC along the δ axis does not imply $\mathrm{NC}^k \neq \mathrm{NC}^{k+1}$, because the relationship between NC^k and N^k is not known. However, it is argued that the δ axis provides a natural subdivision of NC. To support this, a careful analogy with the classes $\mathrm{DTIME}(n^k)$ is made, illuminating a surprisingly close relationship between PTIME and NC. The definition of N^k uses ramified constructs related to a combination of parallel time and space.

1. Discussion: the δ axis

Uniform NC is conventionally divided into classes NC^k based on circuit depth, i.e., parallel computation time. But the hierarchy N^k defined here also provides a way of dividing up NC into subclasses: uniform $\mathrm{NC} = \bigcup_k \mathrm{N}^k$. The division of NC into levels N^k is called the δ *axis*, in contrast to the time axis used to define the levels NC^k. Lemmas 5.2 and 5.4 suggest that the δ axis incorporates elements of both time and space complexity, rather than time alone.

The traditional view is that the most important measure of circuit complexity is the depth of the circuit, i.e. the parallel computation time. Circuit size is relegated to a distinctly secondary status in definitions of NC^k, reflected by the fact that size is

* Corresponding author.

E-mail addresses: sjb@cs.utoronto.ca (S. Bellantoni), isarocha@ptmat.lmc.fc.ul.pt (I. Oitavem).

[1] The author thanks the University of Toronto and Universidade de Lisboa for their support during this research.

[2] The Grant PRAXIS XXI/BD/11217/97 from "Fundação para a Ciência e para a Tecnologia" is deeply acknowledged. The author also would like to thank to the University of Toronto, Universidade Nova de Lisboa and CMAF.

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2003.10.021

only controlled by saying it is bounded by any polynomial. The exponent of the size polynomial is entirely ignored by traditional definitions of NC, while all attention is given to the exponent of the polynomial-log depth of the circuit. Yet circuit size is clearly a very important consideration in building an actual circuit: it is limited by heat dissipation rates and wire congestion. Of course circuit size has been investigated by numerous authors; but it does not appear in the basic definition of the NC^k hierarchy (except perhaps indirectly in the uniformity condition). We argue that the definition of the δ axis, by incorporating the exponent of the size-bounding polynomial as well as the exponent of the log-polynomial depth, is an appropriate single measure of circuit complexity.

To support this argument we adduce three lines of evidence.

The first is an analogy between the levels N^k along the δ axis and the levels $\text{DTIME}(n^k) \subset \text{PTIME}$ of deterministic polynomial time computation. While some may feel that PTIME is a "simpler" class than NC, the analogy described here shows that the definitional structure of NC is essentially similar to that of PTIME. To demonstrate the analogy we provide a term system P defined quite analogously to N, and with a nearly identical δ measure, such that P^k is equivalent to $\text{DTIME}(n^k)$ while P^k is quite analogous to N^k. The classes N^k also share some major properties of $\text{DTIME}(n^k)$. Specifically, the classes N^k form a hierarchy which strictly separates NC, and this result is proved by diagonalization much as in the case of $\text{DTIME}(n^k)$ and PTIME. Not all the similarities between N^k and $\text{DTIME}(n^k)$ are of such copacetic properties. The failure to be closed under arbitrary composition is an obvious undesirable property shared by both $\text{DTIME}(n^k)$ and N^k. Neither is there an abundance of specific lower bounds in either system. Another "negative" analogy is a mild lack of robustness under varying models of computation. For example, time $O(n^{k+1})$ on one-tape Turing machines is not exactly the same as time $O(n^k)$ on multi-tape Turing machines. Likewise, it is not evident that defining N^k as we have, by a term system and δ measure, is the same as defining $N^k_* = \{f : f$ computable in ATM time $O(\log^{k_1} n)$ and space $k_2 \log n + O(1)$ with $k_1 + k_2 = k\}$. Indeed there are any number of ways of combining space and time measures into a single axis, just as there are any number of ways of defining deterministic time computation. It is a thesis, not a provable fact, that all reasonable ways of doing so will be roughly interchangeable. It would be of interest to examine, for example, the relationship between the δ axis and Immerman's FO expressibility characterization [10] of NC, using a combined measure of number-of-variables and iteration depth. Our argument here is that the δ axis provides one reasonable way of simultaneously incorporating time and space bounds.

The second line of evidence for the naturalness of the δ axis, is that lower bound results can be obtained. The fact that $N^k \neq N^{k+1}$ can be proved easily for all k, is a concrete reason to think that the N^k hierarchy is mathematically malleable. Separability along the δ axis gives more meaning to statements such as "function f is in level k of the hierarchy". One may also include here the opinion that the definition of N^k is ontologically simple in that it does not require subsidiary notions of deterministic logspace uniformity, wires, work tapes, bounding functions, etc.

The third line of evidence in favour of the δ axis is that it categorizes specific mathematical functions in a way that usefully corresponds to our intuition of complexity.

Class N^0 is very weak, certainly weaker than AC^0. Parity and OR are not in N^0, but are in N^1. Addition is in N^2. It is open to prove that addition is not in N^1. One thinks of N^1 functions as "local" in the sense that to determine the importance of a given bit of the input one does not have to examine any other bits of the input. In contrast, addition (counting) intuitively seems to require that one access more than one bit at a time in order to determine the result.

Ways of counting recursions have often been introduced to provide a gradation of recursive functions, for example by Parsons [16] for unramified recursions and [4,2,13,14] for ramified recursions. This paper also draws on work of Leivant [11] which used the "nesting depth of recursions" in a ramified context to delineate $\text{DTIME}(n^k)$. The characterization of NC used here has some analogies with the characterization of NC reported by Leivant in [12]. However, the recursion scheme used here is more restricted in that it does not allow arbitrary parameter substitution. On the other hand the recursion used here is more general in that the substituted values are put into tier 1 rather than tier 0. Thus, the recursion scheme is different from that of Leivant, *prima facia* both weaker and stronger. Moreover, the proofs in the current paper take in account the fact that during a composition $f(g(x))$ the length of $g(x)$ may be superpolynomial in the length of x, necessitating excessively large query strings in the construction. This problem is resolved by use of a node numbering scheme. For early work in recursion and parallel complexity see [1,6,7,20].

2. Term system

Let us consider the free algebra $\mathbb{T}$, which has three constructors: $\mathbf{0}$, $\mathbf{1}$ and $*$, of arities 0, 0 and 2, respectively. Usually, $\mathbb{T}$ is interpreted over the set of all finite binary trees and infix notation is used for $*$. For instance, $(0*1)*(0*1)$ is interpreted by the tree

We define a term system N^*, based on the tree algebra $\mathbb{T}$, as follows: N^* is the set of lambda expressions of type level at most 1 formed by lambda abstraction and application using ground-type variables V ranging over $\mathbb{T}$, the "initial" constant symbols $C = \{\mathbf{0}, \mathbf{1}, *, \mathbf{l}, \mathbf{r}, \mathbf{c}\}$, and recursion constant symbol $\mathscr{R}$. We use infix notation for $*$; for other constants one assumes association to the left. The free variables of a term t are $FV(t)$.

Each constant symbol has the arity and type implied by the expressions used below, with the left and right sides of each conversion rule having ground type:

$$\mathbf{l}\,\mathbf{0} \mapsto \mathbf{0}$$
$$\mathbf{l}\,\mathbf{1} \mapsto \mathbf{1}$$
$$\mathbf{l}\,(u*v) \mapsto u$$
$$\mathbf{r}\,\mathbf{0} \mapsto \mathbf{0}$$

$$\mathbf{r}\,\mathbf{1} \mapsto \mathbf{1}$$
$$\mathbf{r}\,(u{*}v) \mapsto v$$
$$\mathbf{c}\,\mathbf{0}\,x\,y\,z \mapsto x$$
$$\mathbf{c}\,\mathbf{1}\,x\,y\,z \mapsto y$$
$$\mathbf{c}\,(u{*}v)\,x\,y\,z \mapsto z$$
$$\mathscr{R}\,g\,h\,p\,\mathbf{0} \mapsto g\,p\,\mathbf{0}$$
$$\mathscr{R}\,g\,h\,p\,\mathbf{1} \mapsto g\,p\,\mathbf{1}$$
$$\mathscr{R}\,g\,h\,p\,(u{*}v) \mapsto h\,p\,(u{*}v)\,(\mathscr{R}\,g\,h\,(p{*}\mathbf{0})\,u)\,(\mathscr{R}\,g\,h\,(p{*}\mathbf{1})\,v)$$

Of course, we also allow β reductions: $(\lambda x\,r)s \mapsto r[s/x]$.

Notice that **l** and **r** denote, respectively, the "left" and the "right" destructor of the tree algebra $\mathbb{T}$, and **c** denotes the conditional.

Attention is restricted to type level at most 1 expressions, and by "term" in the following one means such an expression. These are exactly the expressions such that $\mathscr{R}$ only occurs in a context $\mathscr{R}\,g\,h$ for terms g and h. Thus, every term has one of the following forms: an element of $C \cup V$; or $(\lambda x\,t)$; or $((\mathscr{R}\,g)\,h)$; or $(r\,s)$ with s of ground type. In writing these cases below and referring to the general form $(r\,s)$, one means that s is of ground type.

When defining a term using recursion, one usually writes it informally using equations rather than explicitly using the constant $\mathscr{R}$. Extra parameters in such equations are free variables in the $\mathscr{R}$ term. Likewise one may use equations to define e.g. $f\,x = g(h(x))$, meaning that f is $\lambda x.g(h(x))$.

The use of the "path information" p in recursion is justified as follows. Consider an ordinary recursion on notation, such as $f(0) = g$ and $f(x) = h(x, f(\lfloor x/2 \rfloor))$. The subwords $w \in \{x, \lfloor x/2 \rfloor, \lfloor \lfloor x/2 \rfloor /2 \rfloor, \ldots\}$ encountered during the recursion are uniquely identified by $|w|$, because all of these subwords start at the high-order bit of x. Thus, given the subword w itself, one also knows *which* subword of x it is, namely the subword located at the high-order end of x. But now consider a recursion according to $\mathscr{R}$ above. A subtree w encountered during such a recursion could be located anywhere in x—the value of w itself does not uniquely identify which subtree is under consideration. To uniquely identify the subtree being considered at the current stage of the recursion, one also requires path information such as p. For instance, if f is defined by recursion on x of the form

then the input

will occur during the recursion process. However, this value by itself does not uniquely identify a subtree of the input x; in fact there are two subtrees in x having this value. One does not know whether the $0 * 1$ subtree is the left or the right subtree of x. That

missing information is given by the path. One uses $p*\mathbf{0}$ for a left branching and $p*\mathbf{1}$ for a right branching.

A few definitions are made precise in Appendix A. If the reader is not concerned with details then the following English may suffice: "balanced" trees have a number of leaves $|x| = 2^{\lceil x \rceil}$ each labeled zero or one; "paths" are skewed trees headed by a "0" with binary digits listed down one leg of the tree; paths are concatenated by $\oplus$, and by $x \dot{-} p$ one means the subtree of x reached by following path p from the root.

3. Rank and degree of terms

We introduce a measure of rank, $\hat{\rho}(t)$, which is intended to count the number of "impredicative" recursions. The rank $\hat{\rho}$, differs from the rank measure ρ used in [4]. The earlier ranking ρ was intended to reflect the compositional scheme of [3], in which e.g. $d(d(x))$ might count as rank 1 even if d is defined by recursion on x. However, $\hat{\rho}$ defined here is intended to reflect the more strict compositional requirements typically used in the tiering systems of Leivant [11]. In such a scheme, recursively-defined values are always assigned a strictly lower tier than the recursion input. Thus, under $\hat{\rho}$ one assigns rank at least 2 for x in $d(d(x))$, if d is defined by a recursion on x. However, this does not imply that set of functions delimited by $\hat{\rho}$ is weaker than that defined by ρ. For instance, if $\rho(d(d(x))) \leqslant 1$ then there is another term t computing the same function and such that $\hat{\rho}(t) \leqslant 1$.

While N^* consists of the primitive recursive functions over trees, the 0/1 valued terms having rank less or equal than 1 will be shown to be equivalent to NC. Thus, in the subsequent part of the paper, attention is restricted to the following class:

Definition 3.1. $\mathrm{N} = \{t \in \mathrm{N}^* : \hat{\rho}(t) \leqslant 1\}$.

In this section we describe the class N informally as a subset of N^* defined by two ramified types. Nevertheless, in the proofs we use the formal definition of $\hat{\rho}$ provided in Appendix A, which categorizes the entire recursive system N^*.

One introduces types $\mathbb{T}_k$ for $k \in \{0,1\}$, both having the same extension as $\mathbb{T}$ but differing intensionally by the tier, k. Likewise, for each k one has ground variables V_k of type $\mathbb{T}_k$ and initial functions $0_k, 1_k, *_k, \mathbf{l}_k, \mathbf{r}_k, \mathbf{c}_k$ defined over $\mathbb{T}_k$. Additionally one includes the identity function π of type $\mathbb{T}_1 \to \mathbb{T}_0$. For the recursor $\mathscr{R}$, one assigns types such that each term $\mathscr{R}_k ghpn$ has type $\mathbb{T}_0$ while p and n have type $\mathbb{T}_1$. One requires that in each reduction rule, the redex and reduct have the same tier. This defines a system similar to those well known from the work of Leivant [11] and others.

To understand $\hat{\rho}(t)$, one maps the unramified term t to a ramified term t' having the same structure as t except for the minimal interposition of π as required. If the mapping can be defined using the two-tiered function symbols and preserving typed composition, then $\hat{\rho}(t) \leqslant 1$ and $t \in \mathrm{N}$. Again, this discussion is provided only for intuition; please refer to Appendix A for the formal definition of $\hat{\rho}$ and to see how it ranks all unramified terms. Incidently, this mapping approach does not generalize to more than two tiers; see [15, p. 26].

A salient feature of N is that values used as the pattern in a recursion, are either inputs or are built from inputs by applying a few constant functions. One cannot recurse on a value that itself was produced from inputs by another recursion. Indeed this feature is the main purpose of $\hat{\rho}$. These ideas are made precise in lemma A.1 in Appendix A, which shows that in a term of the form $\mathscr{R}\, g\, h\, p\, m$, the subterms p and m are $\mathscr{R}$-free.

In the next section we prove that N is equivalent to NC.

However, our main contribution is not simply to re-characterize NC, but to introduce and analyse the measure δ given next.

Measure δ is defined on terms by $\delta(t) = \delta(\mathrm{nf}_\beta(t))$ where

$$\begin{aligned}\delta(\lambda\vec{x}.D\vec{u}) &= \max\{\delta(u_i) : i \geqslant 0\} \quad \text{for } D \in C \cup V,\\ \delta(\lambda\vec{x}.\mathscr{R}\, g\, h\, \vec{u}) &= \max(\{1+\delta(h), 1+\delta(g)\} \cup \{\delta(u_i) : i \geqslant 0\}).\end{aligned}$$

Consider a term $(r\, s)$, with s of ground type as usual. One has $\delta(r\, s) = \delta(\mathrm{nf}_\beta(\mathrm{nf}_\beta(r)\, \mathrm{nf}_\beta(s)))$. The only possible β redex in $\mathrm{nf}_\beta(r)\mathrm{nf}_\beta(s)$ is one of the form $(\lambda x.r')\mathrm{nf}_\beta(s)$; and with s of ground type this implies $\mathrm{nf}_\beta(\mathrm{nf}_\beta(r)\mathrm{nf}_\beta(s)) = r'[x \leftarrow \mathrm{nf}_\beta(s)]$. At worst, during this substitution the $\delta(s)$ nested occurrences of $\mathscr{R}$ in $\mathrm{nf}_\beta(s)$ are moved inside of $\delta(r)$ nested occurrences of $\mathscr{R}$ in r'. One concludes $\delta(r\, s) \leqslant \delta(r) + \delta(s)$.

A simple induction on the structure of terms shows that the δ degree of a term is at least as great as δ of any subterm. Therefore, if s is a subterm of t and $\delta(t) \leqslant k$ then $\delta(s) \leqslant k$.

Definition 3.2. Define

$$\mathrm{N}^k = \{t \in N : \delta(t) \leqslant k\}$$

A few examples may clarify these definitions. Consider defining

$$\begin{aligned}0\hat{\oplus}z &= z,\\ 1\hat{\oplus}z &= z,\\ (x*y)\hat{\oplus}z &= (x\hat{\oplus}z)*(y\hat{\oplus}z),\\ 0\hat{\otimes}z &= 1,\\ 1\hat{\otimes}z &= 1,\\ (x*y)\hat{\otimes}z &= z\hat{\oplus}((x\hat{\otimes}z)*(y\hat{\otimes}z)),\\ \hat{e}(0) &= 1,\\ \hat{e}(1) &= 1,\\ \hat{e}(x*y) &= \hat{e}(x)\hat{\oplus}\hat{e}(y).\end{aligned}$$

One has $\hat{\rho}(\hat{\oplus}) = 1$, essentially because $\hat{\rho}(*) = 0$. The second input z of $x\hat{\oplus}z$ has rank 0, and this is the critical position used in the recursion defining $\hat{\otimes}$. One then gets $\hat{\rho}(\hat{\otimes}; 1) = 1 + \hat{\rho}(\hat{\oplus}; 2) = 1$ and $\hat{\rho}(\hat{\otimes}; 2) = \hat{\rho}(\hat{\oplus}; 1)$, leading to $\rho(\hat{\otimes}) = 1$. On the other hand, $\rho(\hat{e}) = 2$ due to the fact that $\hat{e}$ is defined by a recursion through the rank 1 input of $\hat{\oplus}$.

An example of the use of path information is the following *double recursion* derived rule: given g and h, one may define f such that for all isomorphic trees $x*y$ and $a*b$,

$$f\,i\,j = g\,i\,j \quad \text{for } i,j \in \{0,1\},$$
$$f\,(x*y)(a*b) = h(x*y)\,(a*b)\,(f\,x\,a)\,(f\,y\,b).$$

Recalling $\varepsilon = 0$, it is accomplished by defining

$$f'\,0\,q\,x\,a = g\,(x \dot{-} q)\,(a \dot{-} q),$$
$$f'\,1\,q\,x\,a = g\,(x \dot{-} q)\,(a \dot{-} q),$$
$$f'\,(u*v)\,q\,x\,a = h\,(x \dot{-} q)\,(a \dot{-} q)\,(f'\,u\,q*0\;xa)\,(f'\,v\,q*1\,x\,a),$$
$$f\,x\,a = f'\,x\,\varepsilon\,x\,a.$$

Equivalence follows by an induction proving $f\,(x \dot{-} q)(a \dot{-} q) = f'\,(x \dot{-} q)\,q\,x\,a$ under the hypothesis that x and a are isomorphic. Base and step functions are implicit in these recursive equations. These may have δ as much as $1 + \delta(g)$ and $1 + \delta(h)$, because a recursion is used to define $z \dot{-} p$.

4. Definition of NC

Background on a few common characterizations of NC can be summarized as follows. Notes on computation models: A *language* is a boolean function on integers. By NC^k and AC^k one means languages accepted by uniform boolean circuit families of depth $\mathrm{O}(\log^k n)$ and size $2^{\mathrm{O}(\log n)}$ with bounded (for NC^k) or unbounded (for AC^k) gates. Uniformity means FO uniformity. An integer is input to a circuit family by coding it as a minimal-length binary string and selecting the circuit for inputs of that length. One has $\mathrm{NC} = \bigcup_k \mathrm{NC}^k = \bigcup_k \mathrm{AC}^k = \mathrm{AC}$. Alternating Turing Machines (ATMs) are assumed to have one work tape; integers are input as finite oracles which query bits of the binary representation of the integer. Oracles, universal, and existential states may diverge, indicated by the "undetermined value", $\perp$; one understands $0 \vee \perp = \perp$, $1 \vee \perp = 1$, $0 \wedge \perp = 0$, $1 \wedge \perp = \perp$. Concurrent Read Concurrent Write Parallel Random Access Machines (CRCW PRAMs) are assumed to have lowest-processor priority write resolution rule and unit-cost local operations $+$, $-$, $=$, $\leqslant$ together with memory access and indexing operations. As well, for $k \geqslant 1$ one may allow the unit-cost operations $x \dot{-} p = \lfloor x/2^{\lceil \log(p+1) \rceil} \rfloor$ and $x \oplus p = x(2^{\lceil \log(p+1) \rceil}) + p$. The exact placement of the input into shared memory cells is not critical.

Theorem 4.1. *Let $k \geqslant 1$. For any language $L \subseteq \{0,1\}^*$:*

(1) *L is in uniform AC^k iff it is recognized by an ATM in $\mathrm{O}(\log^k n)$ alternations and $\mathrm{O}(\log n)$ space.*
(2) *L is recognized by an ATM in $\mathrm{O}(\log^k n)$ alternations and $\mathrm{O}(\log n)$ space iff it is recognized by a CRCW PRAM in $\mathrm{O}(\log^k n)$ time and $2^{\mathrm{O}(\log n)}$ processors.*
(3) *L is recognized by a CRCW PRAM in $\mathrm{O}(\log^k n)$ time and $n^{\mathrm{O}(1)}$ processors iff it is expressed by a first-order formula iterated $\mathrm{O}(\log^k n)$ times.*

(4) *L is recognized by an ATM in* $O(\log^k n)$ *time and* $O(\log n)$ *space iff it is in* (*uniform*) NC^k.
(5) $NC^k \subseteq AC^k \subseteq NC^{k+1}$.
(6) *If L is recognized by an CRCW PRAM in* $O(\log^k n)$ *time and* $n^{O(1)}$ *processors, then L is in* NC^{k+1}.

Proof. (4) was proved by Ruzzo [17]. Ruzzo's proofs of Theorems 3 and 4 imply (1) under the correspondence of arbitrary fan-in gates as a circuit of nested fan-in two gates. (2) is due to Ruzzo and Tompa; a proof appears in [19]. For (3), refer to [10]. Item (5) is well known. For (6), Stockmeyer and Vishkin [19] showed that the CRCW PRAM using the given resources can be simulated in AC^k; this is in NC^{k+1}

The definitions in the various cited papers are all compatible. Immerman showed that FO-uniformity is the same as U_E uniformity for $k \geqslant 1$, and noted that his "CRAM" model is equivalent to CRCW PRAMs for classes at and above time $\log n$. The $\dot{-}$ and $\oplus$ operations are easily implementable using his Shift operation. Concerning the placement of inputs in the memory cells, see [10, Corollary 3.4]. Ruzzo does not explicitly state that the ATM has only one tape, as Stockmeyer and Vishkin do, but only one tape is required for his proof of Theorem 4. □

N-terms are defined over trees, not over integers. As well, NC formulated above consists of boolean functions. For present purposes, these differences are most easily resolved by treating terms as a model of computation. An integer is input to a term by coding it as a minimal-size perfectly balanced tree, with the leaves representing the integer in binary (with high-order zeros used to pad the leftmost branches of the tree). An output consisting of just a leaf 1 is interpreted as "true" or "accept"; any other output tree is interpreted as "false" or "reject". In this way, each term defines a language. One sometimes refers to a term as if it were the language that it computes.

5. Relationship of N and NC

Two derived rules illustrate the strength of N.

For the *base extension* derived rule, let f be an N-term of arity 2 and let a and b be natural numbers. Then another N-term $f_{a,b}$ of arity 1 can be constructed, such that if x denotes a perfectly balanced tree, then $f_{a,b}x$ denotes the tree obtained by replacing each leaf in a perfectly balanced tree of height $a\lceil x\rceil + b$ with the denotation of fpx. Here p is the minimal-length path leading to the leaf. Note also that fpx may be defined by recursion on p.

The derived function is defined by $f_{a,b}x = \hat{f}_{a,b}0xx$ where

$$\begin{aligned}
\hat{f}_{0,0}\,pxx &= fpx,\\
\hat{f}_{a+1,0}\,p\,0\,x &= \hat{f}_{a,0}\,pxx,\\
\hat{f}_{a+1,0}\,p\,1\,x &= \hat{f}_{a,0}\,pxx,
\end{aligned}$$

$$\hat{f}_{a+1,0}\, p\,(u{*}v)\,x = (\hat{f}_{a+1,0}\,(p{*}0)\,u\,x){*}(\hat{f}_{a+1,0}(p{*}1)vx),$$
$$\hat{f}_{a,b+1}\, pux = (\hat{f}_{a,b}\,(p{*}0)\,u\,x){*}(\hat{f}_{a,b}\,(p{*}1)\,u\,x).$$

One has $\delta(\hat{f}_{a+1,0}) \leqslant 1 + \delta(\hat{f}_{a,0})$ and $\delta(\hat{f}_{a,b+1}) = \delta(\hat{f}_{a,b})$, leading to $\delta(f_{a,b}) \leqslant a + \delta(f)$.

For the *iteration* derived rule, let a, b and c be natural numbers and F be an N-term of arity 1 such that $\hat{\rho}(F;1) = 0$. Thus, F is *not* defined by recursion on its input, although it may be defined by recursion on some free variables. Another N-term $F^{a,b,c}$ of arity 1 can be constructed, such that if x denotes either a perfectly balanced tree or a path, then $F^{a,b,c}xy$ denotes F iterated $a\lceil x\rceil^b + c$ times starting on y. For example, $t^{1,1,0}z$ is $\lceil z\rceil$ iterations of t. For each a,b,c, let $\underline{a}$ be a path of height a and let $\underline{c}$ be a path of height c. One defines

$$\begin{aligned}
F^{0,b,0}xy &= y,\\
F^{1,0,0}xy &= F\,y,\\
F^{1,1,0}\,0\,y &= y,\\
F^{1,1,0}\,1\,y &= y,\\
F^{1,1,0}\,(u{*}v)\,y &= F\,(F^{1,1,0}u\,y),\\
F^{1,b+1,0}xy &= (F^{1,b,0}\,x)^{1,1,0}\,x\,y \quad \text{for } b > 1,\\
F^{a,b,0}xy &= (F^{1,b,0}\,x)^{1,1,0}\,\underline{a}\,y \quad \text{for } a > 1,\ b \geqslant 0,\\
F^{a,b,c}xy &= F^{1,1,0}\,\underline{c}\,(F^{a,b,0}xy) \quad \text{for } a \geqslant 0,\ b \geqslant 0,\ c > 0.
\end{aligned}$$

Note $\hat{\rho}(F^{a,b,c};2) = 0$; that is, $F^{a,b,c}xy$ is not defined by recursion on y. It follows that $F^{a,b,c}$ is in N. By the definition of δ one has $\delta(F^{0,b,0}) = 0$, $\delta(F^{1,0,0}) = \delta(F)$, and $\delta(F^{1,1,0}) = \delta(F) + 1$. Counting repeated uses of the $\cdot^{1,1,0}$ operation one gets $\delta(F^{1,b,0}) = \delta(F) + b$ and $\delta(F^{a,b,c}) \leqslant \delta(F) + b + 2$.

Lemma 5.1. *Every* NC *language is definable by a closed term of* N.

Proof. In order to simulate ATMs with terms, we need to define how the ATM inputs, binary strings, relate to the trees that are input to terms. Given a binary string X, one defines a minimal perfectly balanced tree x such that $x \dot{-} p_i$ is the ith bit of the binary string, where p_i is the root-to-leaf path representing integer i. Say $t \simeq M$ if $t\vec{x} = M\vec{X}$ under this correspondence, for all binary strings $\vec{X}$. Thus, queries by M become questions of the form "is the root of $x \dot{-} i$ equal to b?" where $b \in \{0,1,*\}$.

It is to be shown that for each $M \in \bigcup_k \mathrm{ATM}(\mathrm{O}(\log^k n), \mathrm{O}(\log n))$ there is a closed term $t \in \mathrm{N}$ such that $t \simeq M$.

Let M be an ATM running in time $T_M = a_0\lceil\vec{x}\rceil^k + a_1$ and space $S_M = b_0\lceil\vec{x}\rceil + b_1$ on input $\vec{X}$, with $\vec{x}$ corresponding to $\vec{X}$ as above.

Each configuration of M is stored in the form $\mathrm{nf}(\mathrm{L}(s,h,d) \oplus w)$, where: s is the tape contents; h is a bit string of the same length as s and containing one 1 at the position of the tape head; d is a bit string, of the same length as s, with 0's in each position of a blank tape and 1's in each position of a nonblank tape cell; and w is the current state of the machine (requiring a constant number m of bits to store). Padding the tape

with blanks, all configurations are the same length. A *configuration tree for time t* is a tree in which each path starting from the root, following the bits in any machine configuration, leads to a subtree either 0, 1, or $0*0$ according as whether the machine halts and rejects, halts and accepts, or does not halt, in t steps when started in that configuration.

One may define a function α which when applied to a configuration, c, and input, $\vec{x}$, returns 0 if c has a rejecting halting state for the input $\vec{x}$, 1 if c has an accepting halting state for the input $\vec{x}$, and $0*0$ otherwise (i.e. c has a universal or existential state). It is defined using a single recursion on c. Then by a use of the base extension derived rule with α, one defines a term CT which on input $\vec{x}$ yields the configuration tree for time 0. Note $\delta(CT) \leqslant d'$ for a small constant d' independent of M.

One may define a term θcT which when given a configuration c and a configuration tree T, returns 0, 1, or $0*0$ according as configuration c is rejecting, accepting, or undetermined, assuming the successor configurations of c have the rejecting, accepting, or undetermined states indicated in T. The definition of θ is by a conditional expression using a few helper functions to access T; these helper functions are each defined by a recursion on c. One has $\delta(\theta) = d'$ for a small constant d'. One gets rank 0 for T in this definition.

By a use of a base extension of height $3(b_0\lceil\vec{x}\rceil + b_1) + m$ on $\Theta := \lambda c.\theta cT$, and then by λT, one gets a term Γ which maps the configuration tree for a time t into the configuration tree for time $t+1$.

Finally, by an iteration of $a_0 \cdot \lceil\vec{x}\rceil^k + a_1$ times on Γ starting on CT, one obtains the configuration tree for time T_M. The output 0 or 1 of the machine is obtained from this using recursion on $\vec{x}$ to find the leaf corresponding to the initial configuration.

One should also analyse δ for these terms. Iteration and base extension both increase δ. Base extensions are used to construct configuration trees whose height depends on the space used by the machine, while an iteration rule is used to simulate the time steps of the machine. One has $\delta(\lambda x.\Gamma^{a_0,k,a_1}x(CT\ x)) \leqslant \delta(\Gamma) + k + 2 + \delta(CT)$. With $\Gamma = \Theta_{2b_0,2b_1+m}$ one has $\delta(\Gamma) \leqslant 3b_0 + \delta(\Theta)$, and since $\delta(\Theta) = \delta(\theta) \leqslant d'$ and $\delta(CT) \leqslant d'$ one gets $\delta(\lambda x.\Gamma^{a_0,k,a_1}x(CT\ x)) \leqslant 3b_0 + 2d' + k + 2$ for a small constant d' independent of M. This is linear in k and b_0. □

Lemma 5.2. *There is a constant d such that for all alternating machines M, letting k and b be such that M runs in time* $O(\log^k n)$ *and space* $b\log n + O(1)$, *there is a term* $t \in N^{d(b+k)+d}$ *defining the same language as M.*

Proof. The analysis of δ is in the proof of the preceding lemma. □

Now consider the other direction, to show that N is not more powerful than NC. First observe that there are terms in N which compute trees of superpolynomial size. For example, $\hat{\oplus}$ and $\hat{\otimes}$ were defined above; these are in N. Observe that $|x\hat{\oplus}a| = |a| \cdot |x|$ and, assuming x is perfectly balanced, $|x\hat{\otimes}v| = |v|^{\lceil x\rceil}$. However, it will happen that these trees contain many identical subtrees. In fact, the computed tree can be coded by a polynomial amount of information—these trees of superpolynomial length are equivalent to rooted directed acyclic graphs (dags) having polynomially many nodes.

For example, $x\hat{\oplus}a$ contains $|x|$ copies of a, which is representable by a dag consisting of x with each leaf replaced by a pointer to a.

Lemma 5.3. *Every closed term $t \in \mathrm{N}$ defines an* NC *language.*

Proof. In fact the proof uses CRCW PRAMs. To define a CRCW PRAM for each term, one proceeds by induction on the length of $\mathrm{nf}_\beta(t)$. Thus, assume t is in β-normal form. The induction hypothesis is strengthened as follows. Let $X_{i_1},\ldots,X_{i_k},A_{j_1},\ldots,A_{j_n}$ be the input values, where $\hat{\rho}(t;i_1)=\cdots=\hat{\rho}(t;i_k)=1$ are the *rank* 1 *input positions* and $\hat{\rho}(t;j_1)=\cdots=\hat{\rho}(t;j_n)=0$ are the *rank* 0 *input positions*. One requires that $\vec{X}$ are trees, but not that they be perfectly balanced. Furthermore $\vec{A}$ may be more general *dags*—rooted directed acyclic graphs with leaves labeled 0, 1 in which each non-terminal node has exactly two successors.

The CRCW PRAM for t will compute a dag representing $t\vec{X}\vec{A}$ in time $\mathrm{O}(\lceil\vec{X}\rceil^{\delta(t)})$, with at most $\mathrm{O}(\#(\vec{X})^{\delta(t)})$ processors. The lemma follows by (6) of Theorem 4.1. Indeed, under a simulation [19] the corresponding circuit has size polynomial in the number of processors; i.e. polynomial in $|\vec{x}|^{\delta(t)}$. By another simulation [17] the space used by a corresponding ATM is big-Oh of the log of this. Altogether there is a constant c, independent of t, such that the space used by the corresponding ATM is at most $c\delta(t)\log|\vec{X}|+c$. The ATM time is bounded by $\mathrm{O}(\log^{\delta(t)}|\vec{X}|)$ as is the PRAM time.

Consider CRCW PRAM memory to consist of "nodes", with each node consisting of three memory cells, the *left*, *right*, and *value* cells. The nodes are numbered starting at 1. Write l_i, r_i, and v_i for the contents of the left, right, and value cells of node i. A dag is coded in memory by setting the left and right cells of nodes so that they contain the numbers of the memory nodes for the left and right sub-dags, if any; or to contain numbers $l_i=b$ and $r_i=0$ where $b\in\{0,1\}$ is a leaf label. Nodes that are not in use are indicated by $l_i=2$ and $r_i=0$. Furthermore, designate extra memory cells to serve as stacks; each entry on each stack is a *pointer*, i.e. a node number indicating the root of a dag. When the machine is started, each processor i has an unused global memory node corresponding to it. Pointers to the inputs are initially on stack number 1; when the machine halts these have been popped and a pointer to the output has been pushed.

If t is $\lambda\vec{z}.\mathbf{c}au_1u_2u_3$, then one first computes $\lambda\vec{z}.a$ on inputs $\vec{X},\vec{A}$. By the induction hypothesis, there is a subroutine for it. Testing the root node of the result, one then computes the appropriate $\lambda\vec{z}.u_j$ by another subroutine call. Cases of **l** and **r** are no more difficult than this. For $*$ one must allocate a new node; this can be accomplished in constant time by reading all nodes simultaneously and using write resolution to choose the first unused node. (Nodes are never de-allocated). All these cases require only constant time more than that used for the subterms. In addition to the processors required for the subcomputations, one requires at most one more processor in order to test the conditional, perform the pointer manipulation, or allocate the new node.

If t is $\lambda\vec{z}.\mathscr{R}\,g\,h\,p\,m$, then one first computes $\lambda\vec{z}.p$ and $\lambda\vec{z}.m$ on inputs $\vec{X},\vec{A}$, obtaining dags $\hat{p}$ and $\hat{m}$ in memory. Algorithms for these are available by the induction hypothesis. By the predicativity lemma, A.1, $\hat{p}$ and $\hat{m}$ are computed in constant time from just $\vec{X}$ using the rules for the non-$\mathscr{R}$ constants above. It follows that in a constant amount

of time, using at most $O(\#(\vec{X}))$ processors, one can convert $\hat{p}$ and $\hat{m}$ into trees rather than dags. This may require making a constant number of copies of $\vec{X}$.

Now consider a 'control' processor s for each node in $\hat{m}$, identified by s being a path leading from the root to the node. One also has a group of 'auxiliary' processors for each control processor. The auxiliary processors in group s use stack number s for input and output; the input contents of this stack will be prepared by control processor s. Processors can determine whether they are control processors, and if so for which node of $\hat{m}$, in time $O(\lceil \hat{m} \rceil)$. Processors can then allocate themselves into auxiliary groups in time $O(\lceil \hat{m} \rceil)$. One only uses auxiliary processors corresponding to unused memory nodes, so each group also has a part of global memory for its own use.

First, each control processor s constructs (pointers to) the trees $p_s = \hat{p} \oplus s$ and $m_s = \hat{m} \dot{-} s$; this requires time $O(\lceil s \rceil) = O(\lceil \hat{m} \rceil)$. Now all the control processors proceed to activate themselves in cycles; on the ith cycle the control processors at distance i from the leaves of $\hat{m}$ will be active. Within each cycle, each active control processor, s, will signal its auxiliary processors to compute a dag, r_s. On the 0th cycle, each s is a path to a leaf node of $\hat{m}$; then r_s is obtained by evaluating $\lambda \vec{z}, a, b.\, g\, a\, b$ on the inputs $\vec{X}, \vec{A}, p_s, m_s$. An algorithm for doing so is available by the induction hypothesis. On subsequent cycles, s is a path to an interior node of $\hat{m}$; then r_s is obtained by evaluating $\lambda \vec{z}, a, b, c, d.\, h\, a\, b\, c\, d$ on inputs $\vec{X}, \vec{A}, p_s, m_s, r_{s*0}, r_{s*1}$. Pointers to the latter values are on top of stacks $s*0$ and $s*1$, as they were calculated on a previous cycle. Again an algorithm for the computation is available by the induction hypothesis.

In the cases of $t = \lambda \vec{z}.D\vec{u}$ with $D \in C \cup V$, running time is a constant plus the running time for the subterms, $O(1) + \sum_i O(\lceil \vec{X} \rceil^{\delta(u_i)})$, which is $O(\lceil \vec{X} \rceil^{\delta(t)})$.

To analyse the running time in the case of $\mathscr{R}$, first observe that p and m are $\mathscr{R}$-free by predicativity A.1. $\mathscr{R}$-free terms are computed by the above proof in constant time (i.e. $\delta(p) = \delta(m) = 0$). It also follows that $\lceil \hat{p} \rceil = O(\lceil \vec{X} \rceil)$ and $\lceil \hat{m} \rceil = O(\lceil \vec{X} \rceil)$, leading to $\lceil p_s \rceil = O(\lceil \vec{X} \rceil)$ and $\lceil m_s \rceil = O(\lceil \vec{X} \rceil)$. Similarly $\#(\hat{p}) = O(\#(\vec{X}))$ and $\#(\hat{m}) = O(\#(\vec{X}))$.

Continuing the time analysis in the case of $\mathscr{R}$, note that the critical terms r_0 and r_1 are rank 0 inputs, hence do not enter into the time or processor bounds. Using the induction hypothesis on h and g, one has a time of $\lceil \hat{m} \rceil \cdot O(\lceil \vec{X}, p_s, m_s \rceil^{\delta(h)})$ for all the computations of h at nodes of $\hat{m}$, and a time of $O(\lceil \vec{X}, p_s, m_s \rceil^{\delta(g)})$ for the computation of g. Observe $\delta(t) = \max\{1 + \delta(h), 1 + \delta(g)\}$. With the estimates on $\hat{m}$, p_s and m_s one has a total time of $O(\lceil \vec{X} \rceil^{\delta(t)})$.

Processor bounds are $O(\#(\vec{X}, p_s, m_s)^{\delta(t)})$ processors for each of $O(\#(\hat{m})) = O(\#\vec{X})$ computations of h, plus $O(\#(\vec{X}, p_s, m_s)^{\delta(t)})$ processors for each of $O(|\hat{m}|) = O(\#(\vec{X}))$ computations of g, plus $O(\#(\hat{p}) + \#(\hat{m})) = O(\#(\vec{X}))$ processors for the expansion of $\hat{p}$ and $\hat{m}$ into trees. With $\#(p_s, m_s) = O(\#(\vec{X}))$, altogether this is $O(\#(\vec{X})(\#(\vec{X})^{\max\{\delta(h), \delta(g)\}}))$ processors, which is $O(\#(\vec{X})^{\delta(t)})$. □

Lemma 5.4. *There is a constant c such that for each term $t \in \mathrm{N}$, there is an ATM computing the same language as t in time $O(\log^{\delta(t)} n)$ and space $c\delta(t) \log n + O(1)$.*

Proof. The analysis was given in the preceding proof. □

6. The analogy of P and N

We constructed N based on the algebra $\mathbb{T}$. The term system P results from adopting the same procedure as before, but starting with a different free algebra—$\mathbb{W}$.

$\mathbb{W}$ is the word algebra generated by ε, $\mathbf{s}_0$ and $\mathbf{s}_1$, of arities 0, 1 and 1 respectively, which is usually interpreted over the set of all finite sequences of 0's and 1's. It has only one destructor, $\mathbf{p}$, and we reuse the symbol $\mathbf{c}$ to denote the conditional of $\mathbb{W}$.

P^* is the set of lambda expressions of type level at most 1 formed by lambda abstraction and application using ground-type variables V ranging over $\mathbb{W}$, the "initial" constant symbols $C = \{\varepsilon, \mathbf{s}_0, \mathbf{s}_1, \mathbf{p}, \mathbf{c}\}$, and recursion constants $\mathscr{R}^n_1, \mathscr{R}^n_2, \ldots, \mathscr{R}^n_n$ of arity $2n+2$ for each $n \geqslant 0$, with the conversion rules

$$
\begin{aligned}
\mathbf{p}\,\varepsilon &\mapsto \varepsilon\\
\mathbf{p}\,(\mathbf{s}_0 x) &\mapsto x\\
\mathbf{p}\,(\mathbf{s}_1 x) &\mapsto x\\
\mathbf{c}\,\varepsilon\, x\, y\, z &\mapsto x\\
\mathbf{c}\,(\mathbf{s}_0 u)\, x\, y\, z &\mapsto y\\
\mathbf{c}\,(\mathbf{s}_1 u)\, x\, y z &\mapsto z\\
\vec{\mathscr{R}}\,\vec{g}\,\vec{h}\; p\,\varepsilon &\mapsto \vec{g}\; p\,\varepsilon\\
\vec{\mathscr{R}}\,\vec{g}\,\vec{h}\; p\,(\mathbf{s}_0 x) &\mapsto \vec{h}\; p\,(\mathbf{s}_0 x)\,(\vec{\mathscr{R}}\,\vec{g}\,\vec{h}\,(\mathbf{s}_1 p)\, x)\\
\vec{\mathscr{R}}\,\vec{g}\,\vec{h}\; p(\mathbf{s}_1 x) &\mapsto \vec{h}\; p\,(\mathbf{s}_1 x)\,(\vec{\mathscr{R}}\,\vec{g}\,\vec{h}\,(\mathbf{s}_1 p)\, x)
\end{aligned}
$$

and of course $(\lambda x\, r)s \mapsto r[s/x]$.

The tree recursion scheme in N is essentially a simultaneous recursion scheme, as $*$ can be used to tie together the simultaneously computed values. For binary words one must state simultaneous recursion explicitly as above.

6.1. More details

Vector notation has been used freely in this definition to indicate simultaneous recursion. Precisely, if $\vec{t}$ is the vector $t_1, t_2, \ldots, t_m$ then $\vec{t}\vec{s}$ is written for the vector $t_1\vec{s}, t_2\vec{s}, \ldots, t_m\vec{s}$. By $\vec{t} \mapsto \vec{r}$ one means $t_i \mapsto r_i$ for each i. Finally, the superscript n on $\mathscr{R}$ is omitted; it is the number of base functions $\vec{g}$, which is equal to the number of step functions $\vec{h}$, while the arity of each g_i is 2 and the arity of each h_i is $2+2n$.

The letters C, V, $\mathscr{R}$, $\mathbf{c}$ are recycled from their earlier usage; hopefully this will not cause confusion. The style used for $\mathscr{R}$, including the redundant path information p, is meant to bring forward the analogy with N.

Each binary string x is identified with a term containing only constant symbols ε, $\mathbf{s}_0$, and $\mathbf{s}_1$. The length $|x|$ is the number of such constant symbols in x, so that $|\varepsilon| = 1$. The height $\lceil x \rceil$ is the number of uses of $\mathbf{s}_0$ or $\mathbf{s}_1$; thus $\lceil x \rceil = |x| - 1$. These definitions coincide with the definitions for paths defined by trees.

Let DTIME(n^k) be the functions over $\{0,1\}^*$ computable by a deterministic Turing machine with multiple work tapes (and one head per tape) in time $O(n^k)$. Each input is on a separate work tape with the head initially positioned at the least-significant (right-hand) end of the tape. The output of the machine is the contents of the jth work tape when the machine halts in state j.

6.2. The analogy

The rank $\hat{\rho}$ and the degree δ are defined over P^* exactly as for N^*, understanding the maximums in the definitions for each $\mathscr{R}_i^n$ to be taken over all $\vec{h}$ and $\vec{g}$.

Let P be the terms t of P^*, having $\hat{\rho}(t) \leqslant 1$. Let $P^k = \{t : t \in P,\ \delta(t) \leqslant k\}$.

The definitions of P and N are very similar, differing mainly in the use of binary words versus binary trees. We shall show that P^k is equivalent to DTIME(n^k). This suggests that N^k is to NC as DTIME(n^k) is to PTIME.

The system P is in some respects similar to that defined by Bellantoni and Cook [3]. However, the compositional rule follows that of Leivant [11]; this differs from [3]. See Cobham [8] for an earlier use of recursion on notation and Simmons [18] for an earlier use of tiering. A survey is provided by Clote [7].

Lemma 6.1. DTIME(n^k) $\subseteq P^k$, *for* $k \geqslant 1$.

Proof. Let M be a Turing machine running in time $T = a_0\lceil\vec{x}\rceil^k + a_1$. Each configuration of M is stored in the form $(\vec{l},\vec{h},\vec{r},s)$, where $\vec{l}$ are the portions of the tape contents to the left of the head positions, $\vec{h}$ are the scanned bits (the "head bits"), $\vec{r}$ are the portions of the tape contents to the right of the heads written in reverse order (i.e. from the right to the left) and s is the current state of the machine. The terms $\mathrm{next}^i_{\mathbf{l}}, \mathrm{next}^i_{\mathbf{h}}, \mathrm{next}^i_{\mathbf{r}}$ and $\mathrm{next}_{\mathbf{s}}$ (resp. the next "left portion" of the ith-tape contents, the next "ith-head bit", the next "right portion" of the ith-tape contents in reverse order and the next state of the machine) are defined on input $(\vec{l},\vec{h},\vec{r},s)$ by a composition of constant functions accordingly to the instructions of the machine M. The initial values of these are all easily computed with $l_i = \mathbf{p}x_i$, $h_i = \mathbf{c}x_i, 0, 1$, $r_i = \varepsilon$, and $s =$ a suitable constant. Now, first using non-nested recursions to identify the maximum-length input, and then using the (simultaneous) iteration derived rule to simulate $a_0\lceil\vec{x}\rceil^k + a_1$ machine steps, and letting j be the final output from $next_{\mathbf{s}}$, one gets the output of M as being the final output from $next^j_{\mathbf{l}}$. □

Lemma 6.2. $P^k \subseteq$ DTIME(n^k), *for* $k \geqslant 0$.

Proof. The proof is by induction on the syntatic length of normal-form terms. One proves that for all $k \geqslant 0$ and for all $t \in P^k$, there exist a multi-tape Turing machine, M_t, and a polynomial q^k_t, of degree at most k, such that for any assignment of numerals $\vec{x}$ to the free variables of t and for any numerals $\vec{y}$ of the same arity as t, one has $M_t[\vec{x}](\vec{y}) = t[\vec{x}](\vec{y})$ and $M_t[\vec{x}](\vec{y})$ halts in at most $q^k_t(|\vec{x}^1|, |\vec{y}^1|) + \max(|\vec{x}^0|, |\vec{y}^0|)$ steps. Here $\vec{x}^0$ and $\vec{y}^0$ are selected from $\vec{x}$ and $\vec{y}$ corresponding to the rank 0 variables and

inputs of t, and $\vec{x}^1$ and $\vec{y}^1$ are selected corresponding to the rank 1 variables and inputs of t.

Initial functions $\mathbf{p}$, $\mathbf{s}_0$, $\mathbf{s}_1$, $\mathbf{c}$ and projections x are computed easily in constant time. The most relevant case occurs when $t = \lambda\vec{y}.\vec{\mathscr{R}}\vec{g}\vec{h}$. For simplicity, let us assume $t = \lambda\vec{y}.\mathscr{R}gh$. Let $M_t[\vec{x}](p, m, \vec{y})$ be the Turing machine which consists of one use of $M_g[\vec{x}](p', \varepsilon, \vec{y})$ followed by $|m|$ applications of $M_h[\vec{x}](p', m', \vec{y}, r')$, where p' extends p and m' correspondingly is a subword of m, and r' indicates the value produced by the preceding computation. Under the induction hypothesis, the running time for M_h only depends weakly on r'; it appears in the max term only. On the other hand, $\hat{\rho}(\mathscr{R}gh; i) \geqslant 1$ for $i \in \{1, 2\}$, so that $|p|$ and $|m|$ appear in the polynomial term rather than the max term. Defining $q_t^k(|\vec{X}_0|, |p|, |m|) = |m| q_h^{k-1}(|\vec{X}_0|, |p| + |m|, |p| + |m|) + q_g^{k-1}(|\vec{X}_0|, |p|)$, where q_h^{k-1} and q_g^{k-1} are polynomials of degree at most $k-1$ given by induction hypothesis, a straightforward induction proves that the time used by subroutines of M_t is at most $q_t^k(|\vec{x}^1|, |p|, |m|, |\vec{y}^1|) + \max\{|\vec{x}^0|, |\vec{y}^0|\}$. At most cm additional more steps are used for control operations, for a constant c. □

7. Diagonalization of N

Each closed expanded normal form term $t \in \mathrm{N}$ will be encoded by a unary value $\mathbf{e}(t)$ below. Let $\underline{\mathbf{x}}$, $\underline{\mathscr{R}}$, $\underline{\mathbf{c}}$, $\underline{\mathbf{l}}$, $\underline{\mathbf{r}}$, $\underline{*}$, $\underline{\mathbf{0}}$, $\underline{\mathbf{1}}$, and $\underline{\perp}$ be constant numerals.

Given terms $j_1, \ldots, j_k$, define $\langle j_1, \ldots, j_k \rangle := \langle j_1, \ldots, j_{k-1} \rangle * j_k$, with $\langle\rangle := 0$. Given an integer i, define $\underline{i} = \langle 1, \ldots, 1 \rangle$ where there are i occurrences of 1 in the vector.

For terms $n_1, \ldots, n_k$ define $\langle\langle \vec{n} \rangle\rangle = \langle\langle n_1, \underline{1} \rangle\rangle, \ldots, \langle n_k, \underline{k} \rangle\rangle$. One may define a term $\cdot[\cdot]$ with two uses of $\mathscr{R}$, such that $\langle\langle \vec{n} \rangle\rangle[\underline{i}] = n_i$ for all terms $\vec{n}$ and integers i. Using the fact that $\underline{k}$ is in unary, $\underline{k+1}$ can be obtained from $\underline{k}$ with no uses of recursion. Then one also may define a term $\cdot : \cdot$ with zero uses of $\mathscr{R}$, whereby $\langle\langle \vec{n} \rangle\rangle : m$ reduces to $\langle\langle \vec{n}, m \rangle\rangle$.

Considering closed expanded normal-form terms, define

$$\mathbf{e}(\lambda\vec{x}.\mathbf{l}u) = \underline{\mathbf{l}} * \mathbf{e}(\lambda\vec{x}.u),$$

$$\mathbf{e}(\lambda\vec{x}.\mathbf{r}u) = \underline{\mathbf{r}} * \mathbf{e}(\lambda\vec{x}.u),$$

$$\mathbf{e}(\lambda\vec{x}.u_1 * u_2) = (\underline{*} * \mathbf{e}(\lambda\vec{x}.u_1)) * (\underline{*} * \mathbf{e}(\lambda\vec{x}.u_2)),$$

$$\mathbf{e}(\lambda\vec{x}.\mathbf{0}) = \underline{\mathbf{0}},$$

$$\mathbf{e}(\lambda\vec{x}.\mathbf{1}) = \underline{\mathbf{1}},$$

$$\mathbf{e}(\lambda x_{i_1}, \ldots, x_{i_k}.x_{i_j}) = \underline{\mathbf{x}} * \underline{j},$$

$$\mathbf{e}(\lambda\vec{x}.\mathbf{c}tuvw) = ((((\underline{\mathbf{c}} * \mathbf{e}(\lambda\vec{x}.t)) * (\underline{\mathbf{c}} * \mathbf{e}(\lambda\vec{x}.u))) * (\underline{\mathbf{c}} * \mathbf{e}(\lambda\vec{x}.v))) * (\underline{\mathbf{c}} * \mathbf{e}(\lambda\vec{x}.w))) * \underline{\perp},$$

$$\mathbf{e}(\lambda\vec{x}.\mathscr{R}gh\,pm) = ((((\underline{\mathscr{R}} * \mathbf{e}(\lambda\vec{x}.g)) * ((\underline{\mathscr{R}} * \mathbf{e}(\lambda\vec{x}.h))) * ((\underline{\mathscr{R}} * \mathbf{e}(\lambda\vec{x}.p))) * ((\underline{\mathscr{R}} * \mathbf{e}(\lambda\vec{x}.m))) * \underline{\perp}.$$

Interpreters for this encoding (i.e. universal machines) may be defined as follows. One may imagine $e = \mathbf{e}(t)$ and $k = \delta(t)$ in the following:

$$U_k(e, \langle\langle \vec{n} \rangle\rangle) := \begin{cases} \mathbf{l}\, U_k(\mathbf{e}(\lambda\vec{x}.u), \langle\langle \vec{n} \rangle\rangle) & \text{if } e = \mathbf{e}(\lambda\vec{x}.\mathbf{l}u), \\ \mathbf{r}\, U_k(\mathbf{e}(\lambda\vec{x}.u), \langle\langle \vec{n} \rangle\rangle) & \text{if } e = \mathbf{e}(\lambda\vec{x}.\mathbf{r}u), \\ U_k(\mathbf{e}(\lambda\vec{x}.u_1), \langle\langle \vec{n} \rangle\rangle) * U_k(\mathbf{e}(\lambda\vec{x}.u_2), \langle\langle \vec{n} \rangle\rangle) & \text{if } e = \mathbf{e}(\lambda\vec{x}.u_1 * u_2), \\ \mathbf{0} & \text{if } e = \mathbf{e}(\lambda\vec{x}.\mathbf{0}), \\ \mathbf{1} & \text{if } e = \mathbf{e}(\lambda\vec{x}.\mathbf{1}), \\ \langle\langle \vec{n} \rangle\rangle[\underline{j}] & \text{if } e = \underline{\mathbf{x}} * \underline{j}, \\ \mathbf{c}\, U_k(\mathbf{e}(\lambda\vec{x}.t), \langle\langle \vec{n} \rangle\rangle) & \\ \quad U_k(\mathbf{e}(\lambda\vec{x}.u), \langle\langle \vec{n} \rangle\rangle) & \\ \quad U_k(\mathbf{e}(\lambda\vec{x}.v), \langle\langle \vec{n} \rangle\rangle) & \\ U_k(\mathbf{e}(\lambda\vec{x}.w), \langle\langle \vec{n} \rangle\rangle) & \text{if } e = \mathbf{e}(\lambda\vec{x}.\mathbf{c}ruvw), \\ \mathscr{R}(\lambda z_1, z_2.U_{k-1}(\mathbf{e}(\lambda\vec{x}.g), \langle\langle \vec{n} \rangle\rangle : z_1 : z_2)) & \\ \quad (\lambda z_1, z_2, z_3, z_4.U_{k-1}(\mathbf{e}(\lambda\vec{x}.h), \langle\langle \vec{n} \rangle\rangle : z_1 : z_2 : z_3 : z_4)) & \\ \quad (U_k(\mathbf{e}(\lambda\vec{x}.p), \langle\langle \vec{n} \rangle\rangle)) & \\ \quad (U_k(\mathbf{e}(\lambda\vec{x}.m), \langle\langle \vec{n} \rangle\rangle)) & \text{if } e = \mathbf{e}(\lambda\vec{x}.\mathscr{R}ghpm). \end{cases}$$

Lemma 7.1. *U_k can be written as a term in N^k, for all $k \geqslant 2$.*

Proof. Proceed by induction on k. Consider U_k. Given a numeral $e := \mathbf{e}(t)$, by a simple composition of $\mathbf{l}$, $\mathbf{r}$ and $\mathbf{c}$ one may determine which, if any, of the cases defining e holds (this requires a suitable choice for $\underline{\mathscr{R}}$, $\underline{\mathbf{c}}$, $\underline{\mathbf{l}}$, $\underline{\mathbf{r}}$, $\underline{*}$, $\underline{\mathbf{1}}$, $\underline{\mathbf{0}}$). Thus, the case statement in the definition of U_k is definable in N^0. One needs to see how to write U_k as a recursion in N^k; specifically, in each case one must define $U_k(e, \langle\langle \vec{n} \rangle\rangle)$ in terms of $U_k(\mathrm{nf}_\beta(\mathbf{l}e), \langle\langle \vec{n} \rangle\rangle)$ and $U_k(\mathrm{nf}_\beta(\mathbf{r}e), \langle\langle \vec{n} \rangle\rangle)$, or using an N^k term. An N^2 term is in fact used to write $\langle\langle \vec{n} \rangle\rangle[\underline{i}]$. In each other case except for $\mathbf{c}$ and $\mathscr{R}$, the definition gives $U_k(e, \langle\langle \vec{n} \rangle\rangle)$ explicitly in terms of $U_k(\mathrm{nf}_\beta(\mathbf{l}e), \langle\langle \vec{n} \rangle\rangle)$ and $U_k(\mathrm{nf}_\beta(\mathbf{r}e), \langle\langle \vec{n} \rangle\rangle)$. Here $\mathrm{nf}_\beta(\mathbf{l}e)$ and $\mathrm{nf}_\beta(\mathbf{r}e)$ just refer to the immediate subterms of e, hence they are input to the two recursively defined values. For example, the first step of a recursion on $\mathbf{e}(\lambda\vec{x}.\mathbf{l}u)$ is to recurse on the subtrees $\underline{\mathbf{l}}$ and $\mathbf{e}(\lambda\vec{x}.u)$; the recursively computed value at $\mathbf{e}(\lambda\vec{x}.u)$ gives $U_k(\mathbf{e}(\lambda\vec{x}.u), \langle\langle \vec{n} \rangle\rangle)$ required to define $U_k(\mathbf{e}(\lambda\vec{x}.\mathbf{l}u), \langle\langle \vec{n} \rangle\rangle)$.

In the case of $\mathscr{R}$, the subterms involving U_{k-1} are terms in N^{k-1} (hence in N^k) by the induction hypothesis on k. Here one has used that: $k \geqslant \delta(\lambda\vec{x}.\mathscr{R}ghpm)$ implies $k - 1 \geqslant \max\{\delta(g), \delta(h)\}$.

The other subterms in the cases of $\mathbf{c}$ and $\mathscr{R}$ are of the form $U_k(\mathrm{nf}_\beta(Qe), \langle\langle \vec{n} \rangle\rangle)$ where Q is a sequence of up to three applications of $\mathbf{l}$ or $\mathbf{r}$. Regarding the definition of $\mathbf{e}(t)$, all such $\mathrm{nf}_\beta(Qe)$ are easily distinguished by the presence of the numeric identifier $\underline{\mathbf{c}}$ or $\underline{\mathscr{R}}$ with a right branch not equal to $\perp$. Thus the recursion referring to the values $U_k(\mathrm{nf}_\beta(Qe), \langle\langle \vec{n} \rangle\rangle)$ for various Q, can be re-written as a recursion in which all such values are accumulated using a $*$ operation and then retrieved if the right branch is $\perp$. This allows one to re-write the expression defining $U_k(e, \langle\langle \vec{n} \rangle\rangle)$ as a recursion on e in N^k. $\square$

Lemma 7.2. *For all numerals* $\vec{n}$ *one has* $nf(U_k(e, \langle\langle\vec{n}\rangle\rangle)) = nf(t\vec{n})$, *provided* $e = \mathbf{e}(t)$, $k \geqslant \delta(t)$ *and* t *is a closed normal-form term* (*with* $t\vec{n}$ *of ground type*).

Proof. It follows by an induction on the length of the expanded normal form of term t, regardless of k. Each case is straightforward; the case of $t := \lambda\vec{x}.\mathscr{R}ghpm$ is shown here.

Using the induction hypothesis on p and m one obtains $\mathrm{nf}(p[\vec{n}/\vec{x}])$ and $\mathrm{nf}(m[\vec{n}/\vec{x}])$. Under the reduction rules for $\mathscr{R}$, in order to evaluate $\mathrm{nf}(t\vec{n})$ one needs to evaluate $\mathrm{nf}((\lambda\vec{x}.g)\vec{n}p'i)$ for $i \in \{\mathbf{0}, \mathbf{1}\}$ and various values of p' depending on $\mathrm{nf}(p[\vec{n}/\vec{x}])$ and $\mathrm{nf}(m[\vec{n}/\vec{x}])$. By the induction hypothesis on $\lambda\vec{x}.g$ applied once for each such p' and i, one has

$$\mathrm{nf}((\lambda z_1, z_2.U_{k-1}(\mathbf{e}(\lambda\vec{x}.g), \langle\langle\vec{n}\rangle\rangle : z_1 : z_2))p'i) = \mathrm{nf}(g[\vec{n}/\vec{x}]p'i).$$

A similar analysis applies to h. The case is finished by applying the reduction rules for $\mathscr{R}$ to the definition of U_k. □

Theorem 7.3. $\mathrm{N}^{k+1} \neq \mathrm{N}^k$, *for* $k \geqslant 1$. *Correspondingly, the set of languages recognized by* N^{k+1} *terms strictly contains the set of languages recognized by* N^k *terms.*

Proof. Consider an enumeration $e_1, e_2, \ldots$ of all encodings of N^k terms. To obtain a contradiction, suppose the list includes all N^{k+1} terms. Let t be $\lambda z \cdot \mathbf{c}U_{k+1}(z, \langle\langle z\rangle\rangle)\mathbf{101}$. It is an N^{k+1} term—let d be such that e_d encodes t. Since U_{k+1} is universal for N^k, one has $\mathrm{nf}(U_{k+1}(e_d, \langle\langle z\rangle\rangle)) = \mathrm{nf}(tz)$ for all numerals z. In particular setting $z = e_d$ gives $\mathrm{nf}(U_{k+1}(e_d, \langle\langle e_d\rangle\rangle) = \mathrm{nf}(te_d) = \mathrm{nf}(\mathbf{c}\, U_{k+1}(e_d, \langle\langle e_d\rangle\rangle)\mathbf{101})) \neq \mathrm{nf}(U_{k+1}(e_d, \langle\langle e_d\rangle\rangle))$, a contradiction. Since t is 0–1 valued, the language of t is not recognized in N^k. □

The proof has been to construct $\delta(U_{k+1}) = k + 1$ and show there is no $t \in \mathrm{N}^k$ such that t computes U_{k+1}.

8. Categorization of common functions

Decomposition of NC along the δ axis would be senseless if it did not categorize common functions in a way that usefully corresponds to our intuitions of complexity.

For this discussion it is simplest to again treat terms as a model of computation by inputing integers as perfectly balanced trees. The output tree always can be viewed as an integer by reading off the bits of the leaves. While not absolutely necessary, in most cases one arranges that the output is a perfectly balanced tree whose size is known from the size of the inputs.

Lemma 8.1. *Parity is in* N^1, *and not in* N^0.

Proof. Assume $t \in \mathrm{N}^0$ is a closed normal-form term computing parity. A contradiction is immediate: t must be $\mathscr{R}$-free because $\mathscr{R}$-terms have degree at least 1 while t does not; but an $\mathscr{R}$-free term cannot compute parity due to the fact that a composition of

a constant number of initial functions can only examine at most a constant number of bits of the input.

Parity is easily computed by a recursion: $P(0)=0,\ P(1)=1,\ P(x*y)=\mathbf{c}(P(x),\mathbf{c}(P(y),0,1,\perp),\mathbf{c}(P(y),1,0,\perp),\perp)$. □

The same proof applies to simple functions such as OR and AND, but not to MAJORITY or ADDITION. This contrasts with the fact that ADDITION is in AC^0.

Lemma 8.2. *Addition is in* N^2.

Proof. One defines functions $A_0(x,a)$, $A_1(x,a)$, $C_0(x,a)$ and $C_1(x,a)$ such that if x and a are perfectly balanced and of the same size, then $A_i(x,a)$ is a perfectly balanced tree denoting the least-significant $|x|$ bits of $x+a+i$ while $C_i(x,a)\in\{0,1\}$ is the carry bit. They are all defined simultaneously using the double recursion derived rule. The step function has δ zero and does not use the access function $z\dot{-}p$. The base function of the double recursion has δ zero but does use the bit access function, leading to δ one for the base function in N used to expand the derived rule. Combining the base and step functions using recursion gives a total δ of two. □

9. Using degree to measure the time axis

Another measure of complexity in NC is given by

$$\Delta(\lambda\vec{x}.D\vec{u})=\begin{cases}\max\{\Delta(u_i):i\geqslant 0\} & \text{if } D\in C\cup V,\\ \max(\{1+\Delta(h),\Delta(g)\}\cup\{\Delta(u_i):i\geqslant 0\}) & \text{if } D \text{ is } \mathscr{R}gh.\end{cases}$$

This is the same as δ except one has only $\Delta(g)$ instead of $1+\Delta(g)$ in the base of recursion. Define then $N^{\Delta\leqslant k}=\{t\in N:\Delta(t)\leqslant k\}$. Analysis of the ATM and PRAM simulations leads to $N^{\Delta\leqslant k-c}\subseteq NC^k\subseteq N^{\Delta\leqslant k+c}$ for a small constant c. Thus, the measure Δ corresponds to conventional notions of parallel computation time. The diagonalization proof given above fails for Δ because, in the definition of U_k for $\mathscr{R}$, it is not sufficient to use U_{k-1} to simulate g.

10. Conclusion

The relationship of NC to P has been explored by finding a detailed structural analogy between the two classes. The class NC has been separated by diagonalizing along the "δ axis", which is a hierarchy $\bigcup_k N^k=NC$ quite analogous to the familiar hierarchy $\bigcup_k \text{DTIME}(n^k)=\text{PTIME}$. Under this analogy, one must accept the consequence that this natural subdivision of NC incorporates a combination of circuit size and depth measures, rather than being solely based on depth as is the case under the reigning definitional paradigm of NC^k.

The proposed subdivision $\bigcup_k N^k$ provides a simple separation of NC, is quite analogous to the highly accepted subdivision of PTIME, and is realistic in incorporating a

charge for circuit size. These results are in striking contrast to the lack of separations and the combinatorial difficulties encountered when analysing NC.

Appendix A.

A.1. Fine points

To formalize a few preliminary definitions, a *tree* is a term defined using only $\mathbf{0}, \mathbf{1}, *$. Denotations may be defined in the usual way so as to correspond with the reduction rules above, and with variables ranging over the ground type, consisting of trees. Two terms are *equivalent* if they denote the same function under every assignment. A term is sometimes used as if it were the function that it denotes, although of course there are in fact many equivalent terms.

Every term $t \in \mathrm{N}^*$ reduces to a unique normal form term by replacement of subterms under the conversion rules. An induction on the length of the normal form shows that for every term t of type level at most 1, one may find an equivalent term in *expanded normal form*, being of the form $\lambda\vec{x}.D\vec{u}$ where D is a constant or variable and $\vec{u}$ are in expanded normal form and $D\vec{u}$ is of ground type. Write $\mathrm{nf}(t)$ for the expanded normal form term found in this way. By $\mathrm{nf}_\beta(t)$ one means the expanded normal form considering β reductions only.

The *length* $|x|$ of a tree x is the number of leaves in it. The *height* $\lceil x \rceil$ is the maximal nesting depth of $*$; thus $\mathbf{1}*(\mathbf{1}*\mathbf{0})$ has height 2, while a leaf has height 0. The *size* $\#(x)$ of a tree x is the number of nodes, i.e. the number of $\mathbf{0}$, $\mathbf{1}$ and $*$, in the tree; observe that for our trees, $\#(x) = 2 \cdot |x| - 1$. One writes $|\vec{x}|$ for the maximum of $|x_i|$, and likewise $\lceil \vec{x} \rceil$ or $\#(\vec{x})$ for the maximum of $\lceil x_i \rceil$ or $\#(x_i)$. These maximums are understood to be 0 if the vector $\vec{x}$ is empty.

A *path* is a tree which is either 0 or else $p*i$ for some path p and $i \in \{\mathbf{0}, \mathbf{1}\}$. Each path is identified with an element of $\{0,1\}^*$ by identifying path 0 with the empty string ε, and identifying path $p*i$ with the binary string $b_{p*i} = s_i(b_p)$ (that is, binary string b_p followed by low-order bit i). As well, path $p*i$ represents an integer $k_{p*i} = 2 \cdot k_p + i$, with path $\mathbf{0}$ representing integer 0. Given integer k, one may treat it if it were the minimal-length path representing k. One may concatenate two paths by the function: $a \oplus \varepsilon = a$; $a \oplus (s_i\, p) = s_i(a \oplus p)$. Next, a path p defines a subtree $x{-}p$ of any tree x by: $x{-}0 = x$ and $x{-}(p*1) = \mathrm{nf}(\mathbf{r}(x{-}p))$ and $x{-}(p*0) = \mathrm{nf}(\mathbf{l}(x{-}p))$. A *root-to-leaf path* for tree x is a minimal-length path p such that $x{-}p$ is a leaf. Three paths p, q and r can be put all together by an *interleaving*: $\mathrm{L}(p,q,r) = ((\mathrm{L}(\mathbf{l}p, \mathbf{l}q, \mathbf{l}r)*\mathbf{r}\,p)*\mathbf{r}q)*\mathbf{r}r$.

A *perfectly balanced tree* is one in which all root-to-leaf paths have exactly the same length.

A.2. Definition of rank

Consider any term t, any integer $1 \leqslant i \leqslant \mathrm{arity}(t)$, and any free variable $x \in FV(t)$. Ranks $\hat{\rho}(t;i)$ and $\hat{\rho}(t;x)$ are to be defined, where in the latter case one intends that x

refer to the name "x" rather than to the value of x. Thus, $\hat{\rho}(t;i)$ will be the rank of the ith input of t, and $\hat{\rho}(t;x)$ will be the rank of free variable x. Note $\hat{\rho}$ is undefined outside this domain.

That the definition of $\hat{\rho}$ has some value as a means of classification, and we provide it in complete generality for future reference. In the subsequent part of the paper, however, attention is restricted to $\hat{\rho}(t)\leqslant 1$.

The definition is in two sections. The first few clauses defining $\hat{\rho}$ are intended primarily to switch back and forth between free variables and inputs. The rank is summed during a composition.

$$\begin{aligned}
\hat{\rho}(\lambda z.r;i) &= \begin{cases} \hat{\rho}(r;z) & \text{if } i=1,\\ \hat{\rho}(r;i-1) & \text{if } i>1,\end{cases}\\
\hat{\rho}(\lambda z.r;x) &= \hat{\rho}(r;x) \text{ if } x \text{ is not } z,\\
\hat{\rho}(r\,s;i) &= \hat{\rho}(r;i+1) \quad \text{for } s \text{ of ground type},\\
\hat{\rho}(r\,s;x) &= \max\begin{cases} \hat{\rho}(r;1)+\hat{\rho}(s;x)\\ \hat{\rho}(r;x)\end{cases} \quad \text{for } s \text{ of ground type.}
\end{aligned}$$

The remaining clauses defining $\hat{\rho}$ specify the way in which $\hat{\rho}$ counts recursions. One forces the rank of p and x in $\mathscr{R}\,g\,h\,p\,x$ to be one more than the rank of the critical positions in h.

$$\begin{aligned}
\hat{\rho}(D;i) &= 0 \quad \text{for } D\in C,\ 1\leqslant i\leqslant \text{arity}(D),\\
\hat{\rho}(x;x) &= 0 \quad \text{for variable } x,\\
\hat{\rho}(\mathscr{R}gh;i) &= \max\begin{cases} \hat{\rho}(h;i),\\ \hat{\rho}(g;i),\\ 1+\hat{\rho}(h;k)\colon 3\leqslant k\leqslant \text{arity}(h),\end{cases}\\
\hat{\rho}(\mathscr{R}gh;x) &= \max\begin{cases} \hat{\rho}(h;x),\\ \hat{\rho}(h;k)+\hat{\rho}(g;x)\colon 3\leqslant k\leqslant \text{arity}(h).\end{cases}
\end{aligned}$$

The definition of $\hat{\rho}(\mathscr{R}gh;i)$ only holds for $i\in\{1,2\}$, as there are only two inputs to $\mathscr{R}gh$. Each of these two input positions has rank at least 1 in $\mathscr{R}gh$. The reference to arity(h) will come in handy in section 6.2; at present arity$(h)=4$. Only the free variables of $\mathscr{R}\,g\,h$ may have rank 0 in $\mathscr{R}\,g\,h\,p\,x$.

Define $\hat{\rho}(t)=\max(\{\hat{\rho}(t;i)\colon 1\leqslant i\leqslant \text{arity}(t)\}\cup\{\hat{\rho}(t;x)\colon x\in FV(t)\}\cup\{0\})$.

A simple induction on the structure of normal-form terms shows that the rank of a normal-form term is at least as great as the rank of any subterm.

Undefined terms are omitted from the maximums above, with the maximums being undefined if they are taken over no terms. For example, $\hat{\rho}(r\,s;x)=\hat{\rho}(r;1)+\hat{\rho}(s;x)$ whenever $x\in FV(s)\backslash FV(r)$, while $\hat{\rho}(r\,s;x)$ is undefined if $x\notin FV(r\,s)$. In the case of all undefined values on the right-hand side, the left-hand side is also undefined; e.g. the definition of $\hat{\rho}(\lambda z.r,x)$ for $x\notin FV(r)$.

Lemma A.1 (Predicativity). *If* $\lambda\vec{z}.\mathcal{R}ghpm$ *is a normal-form* N *term, then* p *and* m *are* $\mathcal{R}$*-free. Furthermore all free variables of* p *or* m *have rank* 1 *in* $\mathcal{R}ghpm$.

Proof. Arguments for m are similar to those for p. First one proves that p is a rank 0 term. Consider $x \in FV(p)$; by definition, $\hat{\rho}(\mathcal{R}ghpm,x) \geqslant \hat{\rho}(\mathcal{R}gh,1)+\hat{\rho}(p,x) \geqslant \max\{1+\hat{\rho}(h,3),1+\hat{\rho}(h,4)\}+\hat{\rho}(p,x) \geqslant 1+\hat{\rho}(p,x)$. If $\hat{\rho}(p,x) \geqslant 1$ then $\hat{\rho}(\lambda\vec{x}.\mathcal{R}ghpm) \geqslant 2$ contradicting that $\lambda\vec{x}.\mathcal{R}ghpm$ is an N term. It follows that $\hat{\rho}(p)=0$. This also proves the second part of the lemma, that any variable x free in p satisfies $\hat{\rho}(\mathcal{R}ghpm,x) \geqslant 1$.

The first part of the lemma is finished by showing that every normal-form ground type rank 0 term in N (such as p) is $\mathcal{R}$-free. To obtain a contradiction, assume there is an outermost occurrence of $\mathcal{R}$ (in p). Since it is outermost and all other constants and variables have only ground type inputs, and the term is of ground type, this occurrence of $\mathcal{R}$ must be in the form $\mathcal{R}g'h'p'm'$. If p' or m' is not closed then this subterm has rank 1 for the same reasons as above. Thus if p' or m' is not closed then $\mathcal{R}g'h'p'm'$ has rank 1, in contradiction of the term (p) having rank 0. Therefore p' and m' are closed. But p' and m' are in normal form; hence they are trees. But then $\mathcal{R}g'h'p'm'$ forms a redex, contradicting that it is in normal form. □

References

[1] B. Allen, Arithmetizing uniform NC, in: Ann. Pure Appl. Logic, Vol. 53, North-Holland, Amsterdam, 1991.

[2] S. Bellantoni, Ranking arithmetic proofs by implicit ramification, Technical report 96-49, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996; available from ftp://www.dimacs.rutgers.edu/pub/dimacs/TechnicalReports/1996/96-49.ps.gz

[3] S. Bellantoni, S. Cook, A new recursion-theoretic characterization of polytime functions, Comput. Complexity 2 (1992) 97–110.

[4] S. Bellantoni, K.-H. Niggl, Ranking primitive recursions: the low Grzegorczyk classes revisited, SIAM J. Comput. 29 (2) (1999) 401–415.

[5] S. Bellantoni, K.-H. Niggl, H. Schwichtenberg, Higher type recursion, ramification and polynomial time, Ann. Pure Appl. Logic 104 (2000) 17–30.

[6] S. Bloch, Functional characterizations of uniform log-depth and polylog-depth circuit families, in: Proc. Seventh Ann. Structure in Complexity Theory Conf., IEEE, 1992.

[7] P. Clote, Computation models and function algebras, in: E. Griffor (Ed.), Handbook of Recursion Theory, North-Holland, Amsterdam, 1997.

[8] A. Cobham, The Intrinsic Computational difficulty of function, in: Y. Bar-Hillel (Ed.), Logic, Methodology and Philosophy of Science, Vol. II, Jerusalem, 1964, North-Holland, Amsterdam, 1965, pp. 24–30.

[9] F. Ferreira, Stockmeyer induction, in: S.R. Buss, P.J. Scott (Eds.), Feasible Mathematics, 1990, pp. 161–179.

[10] N. Immerman, Expressibility and parallel complexity, SIAM J. Comput. 18 (1989) 625–638.

[11] D. Leivant, Ramified recurrence and computational complexity I: word recurrence and poly-time, in: P. Clote, J.B. Remmel (Eds.), Feasible Mathematics, Vol. II, 1995, pp. 320–343.

[12] D. Leivant, A Characterization of NC by Tree Recurrence, Proc. 39th Ann. Symp. on Foundations of Computer Science, FOCS, 1998, IEEE Computer Society, Silver Spring, MD, 1998, pp. 716–724.

[13] K-H. Niggl, Towards the computational comolexity PR^{ω}-terms, Ann. Pure Appl. Logic 75 (1–2) (1995) 153–178.

[14] K-H. Niggl, The μ measure as a Tool for Classifying Computational Complexity, Arch. Math. Logic 39 (2000) 515–539.

[15] I. Oitavem, Classes of computational complexity: implicit characterizations—a study in mathematical logic, Ph.D. Dissertation, Faculdade de Ciências da Universidade de Lisboa, 2001.
[16] C. Parsons, LCF Considered as a programming language, Zeitschr. Math. Logik Grundlagen Math. 14 (1968) 357.
[17] W.L. Ruzzo, On Uniform Circuit Complexity, J. Comput. System Sci. 22 (3) (1981) 365–383.
[18] H. Simmons, The realm of primitive recursion, Arch. Math. Logic 27 (1988) 177–188.
[19] L. Stockmeyer, U. Vishkin, Simulation of parallel random access machines by circuits, SIAM J. Comput. 13 (12) (1984) 409–422.
[20] D. Suciu, V. Tannen, A query language for NC, in: D. Leivant (Ed.), Lecture Notes in Computer Science, Vol. 690, Springer, Berlin, 1994.